**Advanced Development Techniques**

Midterm Assignment

2023/24/1 semester

The main task of the assignment is to develop a CRUD application that is based on a self-made database, while using source version control (git)

The basic requirements for the assignment are detailed below. Every mandatory requirement must be satisfied, otherwise the assignment will not be accepted!

**Basic requirements are as follows:**

* The assignment must be tracked using the **git** version control system from the beginning. Adding git to your project afterwards will not work! (git is not a file upload site!)
* You need to attach a remote repository to your local changes, this must be done on **github.com.**
* The repository’s name on **Github** must be in the following format: **ABC123\_ADT\_2023241** (where ABC123 is the neptun code, ADT is the subject’s name, 2023241 is the current semester)
* The visibility of the repository must be **private.** You will need to add the **lab teacher’s GitHub account** as collaborator to the project. (This process is not automatic; the teacher will need to accept the invitation so it might not show up right away).
* During development after you are done with a segment, you will need to **commit** your changes. **The minimum number of commits is 25.**
* You will need to **push** your changes to your **GitHub repository** for them to show up on GitHub.
* Sharing code between the teacher and the student shall only be done using GitHub. You **can not** submit your code in via E-mail or other methods.

**Basic requirements for the empty solution layout**

* The solution’s name must be **ABC123\_ADT\_2023241 (where ABC123 is the neptun code),** and you will need to create a **Console App using .Net 6!** The name for the Console App must be **ABC123\_ADT\_2023241.Client!**
* You will also need to create the following **projects** inside the **solution:** (The solution’s name needs to be prefixed before the project’s name)
  + **ABC123\_ADT\_2023241.Models** (Class Library);
  + **ABC123\_ADT\_2023241.Logic** (Class Library);
  + **ABC123\_ADT\_2023241.Repository** (Class Library);
  + **ABC123\_ADT\_2023241**.**Endpoint** (ASP.NET Core Empty – HTTPS not needed!);
  + **ABC123\_ADT\_2023241.Test** (Class Library).
* Add the following **solution folders** (Right on solution -> Add) and move the relevant project’s to their appropriate place:
  + **Backend** (Endpoint, Logic, Repository, Test);
  + **Frontend** (Client);
  + **Shared** (Models).
* It is good practice to have the startup projects set for both the backend and the front end when you press the start button.Right click on the **solution** > Properties > Startup Project > Multiple startup projects > Select start for both the EndPoint and the Client (This won’t show up in git, but that’s not the point)
* You also need to setup dependencies for the projects. To add a dependency right click on it > Add > Project Reference. The way that the dependencies should be set up is detailed below. You are not allowed to add any other dependencies between the layers, except for the Client, where you can add any you may need during development. At the final deadline however only the Models can be added to the Client.
  + **Endpoint** needs: **Logic, Repository, Models** dependencies;
  + **Test** needs: **Logic, Repository, Models** dependencies;
  + **Logic** needs**: Repository, Models** Dependencies;
  + **Repository** needs**: Models** dependency;
  + **Client** needs: **Models** dependency;
* After these steps you should create your local git repository.
  + In VS at the bottom right there is a „**Add to source control**” button. If it does not show up you are missing **Git for Windows** és a **GitHub Extension for Visual Studio.** You can install these with the visual studio installer if you select the **Individual components** tab. For Databases you will need the **Data storage and processing Workload.**
  + After **Add to source control** you will need to login with your GitHub account, check the repository’s name (should be the same as your solution’s name), check that the repository is private, then select „**Create and Publish**”.
  + If you create your repository from Visual Studio a **.gitignore** will be generated automatically with the proper contents.
  + Create an **Others** folder in the solution, > right click > Add > Existing Item. Once this is done, move the **.gitignore** file to this folder.
  + Commit often and push every once in a while, (at the end of each day for example), 1 push can upload many commits!
  + You can download your code, unzip and check if it compiles it will be the same as the teacher will get your code!

**Software requirements**

* The code must compile, code that cannot be compiled will not be checked.
* **.NET 6.0** is required and **InMemoryDb** database must be used with **Entity Framework Core**. Other version will not be accepted!
* Use English for class names variable names etc.
* You need to **create 3 tables** which connect to one another (with foreign keys). You need 3 **Model** classes in the Model class Library. One example could be, Brands where each brand can have multiple car Models, and every car model has multiple rent instances. Lookup tables don’t count as individual tables.
* In the model classes you need to use the **foreign keys** and manage them using **Navigation Properties**, with **LazyLoad**, where possible.
* In the **Repository** layer using the **DbContext** class’ **OnModelCreating** method populate the database using sample data.
* The **Logic** layer can only get the **Repository** layer as dependency, using constructor parameters (dependency injection), The Endpoint’s Controllers can only get the Logic using interfaces and as a constructor parameter. The dependency injection is done via **IoC Container**
* Every **Model** needs a correspoinding **Repository** class that has the CRUD methods, **Create, Read, ReadAll, Update, Delete**). The **ReadAll** method returns **IQueryable<T>** to the logic the **DbSets**
* The **Logic** layer also needs to have these CRUD methods, and you also need 3 **non-crud methods in the logic, which use queries with more than 1 table**. The CRUD and the non-crud methods results should be **IEnumerable**<T>.
* In the **Test** project **NUnit** and **Moq** must be used. The Logic should get a mocked database using **Moq**. The unit test’s main purpose is to test the **non-crud** methods in the **Logic**, and to validate the **Create** method’s error handling (like if a name is an empty string, throw an exception). A Create method in the logic layer differs from one in the **Repository** layer such that, the Logic’s **Create** will do error handling, throws exceptions. The Create in the Repository layer just saves the object in the database without validation.
* You will need to create **10 Unit tests**! Pl: 5 non-crud, 3 create, 2 which can be freely selected;
* **All** the tests must pass! If you have 20 all of them must be passing!
* Every model class should have a corresponding Repository class, (e.g: Car  CarRepository) and a Logic class (Car -> CarLogic). One **Logic** class can you multiple **Repositories**, if a query requires data from other repositories.
* The **Endpoint** project only knows the Logic classes and provides a way to interact with them using API Endpoints. Every **Logic** class can have one or more **ApiController**. The **ApiController’s** Action methods should match with the Logic’s methods.
  + HTTP GET  Read, ReadAll;
  + HTTP POST  Create;
  + HTTP PUT  Update;
  + HTTP DELETE  Delete.
* The **Console** app send API requests to the **Endpoint** using **JSON**. The Console app needs to be able to access all the CRUD functionality (All CRUD and non-CRUD methods). Optionally, for easier Console management you can use the **ConsoleMenu**-Simple NuGet package.

**Milestones**

The following milestones are expected, but their completion won’t be checked.

**October 22., 23:59:59**

* GitHub repo created with the specified name!
* The lab teacher added as a collaborator.
* The empty projects are created!
* After the push the previous steps can be checked on GitHub.

**November 5., 23:59:59**

* Model layer contains atleast 3 classes and they have some content.
* Repository layer has the XYZDbContext class (where XYZ is your own DbContext derived class)
* DbSeed populates the database tables with data.

**November 19., 23:59:59**

* Repository layer has atleast 3 classes
* These contain the CRUD methods.

**November 29., 23:59:59**

* Logic layer has the required amount of classes.
* These have the non-crud and crud methods implemented.
* 10/10 Unit tests can be run, and they pass.
* Endpoint app can be built, and it responds to API requests.

**Deadline**: December 6., 23:59:59

**Resubmission Date**: December 13., 23:59:59

Code that is pushed after these deadlines it will not be checked! If your code does meet the check requirements (detailed below) it will be declined, and you will need to correct it until the resubmission date!

Validation:

The code will be checked automatically using GitHub’s own system to do basic checking (build, correct naming etc..). Some files will need to be copied to your project to run the check after every push. These files will be provided later.

Good Luck!